A Cognitive Approach to Classifying Perceived Behaviors

D. Paul Benjamin

Pace University, 1 Pace Plaza, New York, New York 10038, 212-346-1012, benjamin@pace.edu

Damian Lyons

Fordham University, 340 JMH, 441 E. Fordham Rd., Bronx, NY 10458, 718-817-4485, dlyons@fordham.edu

ABSTRACT

This paper describes our work on integrating distributed, concurrent control in a cognitive architecture, and using it to classify perceived behaviors. We are implementing the Robot Schemas (RS) language in Soar. RS is a CSP-type programming language for robotics that controls a hierarchy of concurrently executing schemas. The behavior of every RS schema is defined using port automata. This provides precision to the semantics and also a constructive means of reasoning about the behavior and meaning of schemas. Our implementation uses Soar operators to build, instantiate and connect port automata as needed. Our approach is to use comprehension through generation (similar to NLSoar) to search for ways to construct port automata that model perceived behaviors. The generality of RS permits us to model dynamic, concurrent behaviors. A virtual world (Ogre) is used to test the accuracy of these automata. Soar's chunking mechanism is used to generalize and save these automata. In this way, the robot learns to recognize new behaviors.

Keywords: robot schemas, Soar cognitive architecture, port automata

1. INTRODUCTION

The ADAPT project (**A**daptive **D**ynamics and **A**ctive **P**erception for **T**hought) is a collaboration of three university research groups at Pace University, Brigham Young University, and Fordham University to produce a robot cognitive architecture that integrates the structures designed by linguists and cognitive scientists with those developed by robotics researchers for real-time perception and control. Our goal is to create a new kind of robot architecture capable of robust behavior in unstructured environments, exhibiting problem solving and planning skills, learning from experience, novel methods of perception, comprehension and use of natural language and speech generation.

ADAPT models the world as a network of concurrent schemas, and models perception as a problem solving activity. Schemas are represented using the RS (Robot Schemas) language[9,10], and are activated by spreading activation. RS provides a powerful language for distributed control of concurrent processes. Also, the formal semantics of RS[14] provides the basis for the semantics of ADAPT's use of natural language. We have implemented the RS language in Soar[8], a mature cognitive architecture originally developed at Carnegie-Mellon University and used at a number of universities and companies. Soar's subgoaling and learning capabilities enable ADAPT to manage the complexity of its environment and to learn new schemas from experience.

This cognitive robotic architecture is currently being developed on Pioneer robots in the Pace University Robotics Lab and in the Fordham University Robotics Lab.

In the next section, we give a brief description of ADAPT's method of comprehending its environment and other agents. In the subsequent sections, we focus on the structure and implementation of ADAPT.

More detailed information is available in the ADAPT publications[2,3,4,5]. More papers and videos on ADAPT are available at the Pace University Robotics Lab website.

2. COMPREHENSION THROUGH GENERATION

ADAPT's approach to comprehension of behaviors is based on the work of Green & Lehman[7], who used Soar to model discourse planning in natural language. Their goal was a computational model of discourse, in which Soar would interact with a human using English. In their task, Soar had to do two things: construct an explanation of an incoming utterance and generate an appropriate response. Their approach was for Soar to construct an explanation for an incoming utterance by attempting to generate a similar utterance on its own. Soar would search among possible combinations of goals and expectations to try to generate this utterance, and when it succeeded in matching the utterance it assumed the human's goals and expectations were those it had found. This gave Soar an understanding of the goals of the person it was conversing with, so that Soar could choose appropriate goals for generating a response. Soar also formed a chunk for this search, so that a similar utterance in the future would be understood in one step.

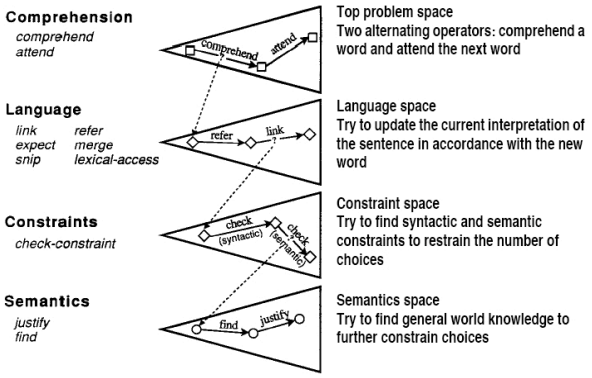

Figure 1. Example goal stack in NLSoar.

For example, if the discourse were about cooking and the incoming utterance were, "Turn the flame down." then Soar would try to generate that same sentence by searching various combinations of goals and knowledge. One combination that works is the knowledge that a high flame cooks food faster and the goal is to cook it slowly. If Soar found this combination, then it would assume the speaker has this same combination of knowledge and goal. This approach requires Soar to have knowledge about cooking, so that it can search that knowledge.

Comprehension through generation is used by ADAPT to understand incoming utterances, just as it is by NLSoar. But ADAPT uses comprehension through generation not only for speech input, but also to understand nonverbal behaviors of other agents in its environment.

Nonverbal behaviors are extremely important to understand, because much of the interaction between agents in the real world is nonverbal. For example, walking alongside a person involves keeping a relatively constant distance from the person, which requires monitoring the person's movements and responding appropriately to them. If the person speeds up or slows down or changes direction, there is probably a reason, and an intelligent robot should understand that reason in order to respond appropriately. If a team is composed of several people and several robots, it is not possible for the people to continually explain their movements to the robots; it is necessary for the robots to be able to generate their own explanations.

ADAPT generates these explanations by using the Ogre3D world model (www.ogre3d.org) to simulate the observed behaviors of other agents. Each agent is represented as a Soar agent that is identical to the robot. ADAPT attempts to replicate the observed behavior by searching through sequences of Soar operator firings for that agent. When an appropriate sequence of Soar operators is found that replicates the observed behavior, ADAPT assumes that the agent has executed this sequence of operators, and attributes to the agent the goals and knowledge used by these operators. ADAPT then constructs an RS port automaton (described in the next section) that generates this sequence of behaviors.

For example, suppose the robot is traveling with a human teammate and the observed behavior is that the human looks ahead and slows down a bit because he sees a vehicle crossing his path and wishes to let it pass. This is a typical behavior that should not need to be explained to the robot. ADAPT can model the human and the vehicle in its 3D world model, and "imagine" the consequences of not slowing down, which will be a collision. This permits ADAPT to infer

the reason for the behavior, and to create a new schema for slowing down to let vehicles pass. This schema is used both for generation (when the robot needs to slow down to permit vehicles to pass) and for comprehension.

The advantage of comprehension through generation is that it provides a deeper understanding of the world than statistical associations can provide. In particular, ADAPT can reason about why complex sequences of actions are performed, and how they can be varied. The disadvantage of this approach is that it is slow; the search for the proper combination of goals and knowledge can be extremely long, and may not be successful. The usefulness of this approach to robotics is not known yet, and will depend on the power of Soar's chunking method.

3. ROBOT SCHEMAS

ADAPT's representation of dynamic behaviors is based on the RS (Robot Schemas) language[14]. RS is a language with a formal model of robot computation that is based on the semantics of networks of *port automata*[10]. A port automaton (PA) is a finite-state automaton equipped with a set of synchronous communication ports[14].

RS builds a network of *sensory-motor schemas* to model the dynamics of both the robot and the environment. A schema is an organized network of actions (abstract and/or concrete), percepts, words, facts and beliefs about some aspect of the world. Schemas also contain explicit qualitative temporal information, e.g. about intervals of time during which an action must take place. The schema relates these components to each other and to other schemas.

For example, the schema for a table would contain various images of tables, and would connect them to facts about tables, such as their typical size, and to actions that typically are performed with tables, such as eating at one (which is another schema). The table schema would be connected to the schemas for chairs, dining rooms, conference rooms and many other relevant concepts.

One of the unique advantages of RS is its formal semantics. Each schema has an associated port automaton that defines the semantics of the schema. ADAPT explicitly constructs this PA for each schema and attaches it to the schema. The PA performs two important functions.

First, it provides the basic semantics for the use of natural language. States in the PA provide the semantics for relations and predicates (adverbs and adjectives). Transitions between the states provide the semantics for verbs.

Second, it enables ADAPT to synthesize hierarchies of schemas by decomposing automata. Benjamin[6] has developed an algebraic method of factoring an automaton's associated semigroup. This method is applied in a straightforward way to PAs, factoring each PA into a collection of simpler PAs and decreasing the cost of perception and action.

A large body of work in intelligent systems employs similar notions of schema. Arbib[1] presents a comprehensive theory of schemas.

3.1 Robot Schemas

ADAPT uses schemas by instantiating them to create schema instances that are connected to each other to form a network. Over time, ADAPT maintains this network, monitoring its behavior and adding schema instances and removing old ones as its goals and the environment change.

A network of processes is typically built to capture a specific robot sensory-motor skill or behavior: sensory processes linked to motor processes by data communication channels and sequenced using *process composition operations*.

For example, the following RS statement defines the i-th joint of a robot in terms of three sub-units, which are connected by connection map that connects similarly named output and input ports:

$$\text{Joint}_i(s)() = [\text{Jpos}_i()(x) \mid \text{Jset}_i(s, x)(u) \mid \text{Jmot}_i(u)()\ ]$$

$\text{Jpos}_i()(x)$ continuously reports the position of joint i on output port x
$\text{Jmot}_i(u)()$ accepts a signal on input port u and applies it to the actuator of joint i
$\text{Jset}_i(s, x)(u)$ accepts a set-point on input port s and iteratively inputs a joint position on port x and outputs a motor signal on port u to drive the joint position to the set-point

This RS schema can then be used as a sub-unit of a guarded move schema that is higher in the hierarchy:

$$\text{Touch}_i = [\text{Tact}_i()(v) \,|\text{Gmove}_i(v)(y) \mid \text{Joint}_i(y)()\,]$$

Tact_i reports on tactile contact on the i-th join on its port v.

Gmove increments the setpoint of the joint actuator as long as it gets a no-contact signal on port v.

Touch_i implements a guarded move of the i-th link.

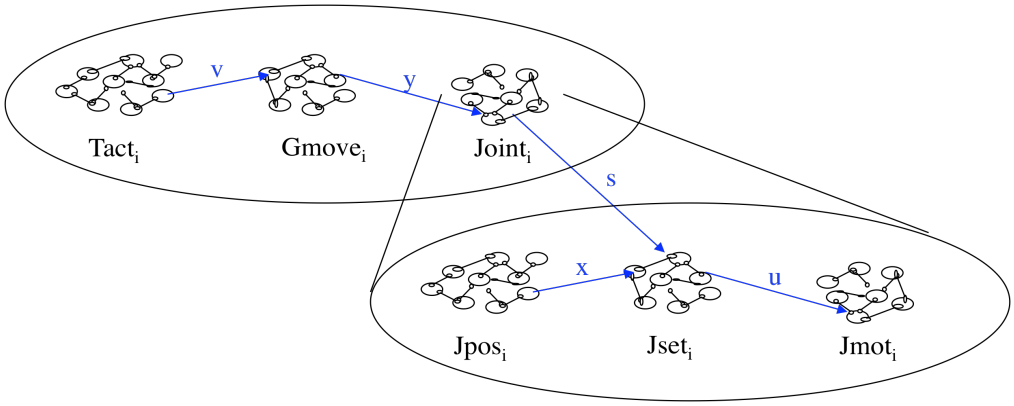

Figure 2. The RS hierarchy for the guarded move schema.

Process Composition in RS is extremely general, encompassing the full range of concurrent process compositions, as is shown in the following table of RS composition operators:

1. Sequential Composition: $\mathtt{T} = \mathtt{P};\mathtt{Q}$. The process $\mathtt{T}$ behaves like the process $\mathtt{P}$ until that terminates, and then behaves like the process $\mathtt{Q}$ (regardless of $\mathtt{P}$'s termination status).
2. Concurrent Composition: $\mathtt{T} = (\mathtt{P} \mid \mathtt{Q})^c$. The process $\mathtt{T}$ behaves like $\mathtt{P}$ and $\mathtt{Q}$ running in parallel and with the input ports of one connected to the output ports of the other as indicated by the port-to-port connection map c. This can also be written as $\mathtt{T} = (\ \Big|_{i \in I}\ \mathtt{P}i\ )^c$ for a set of processes indexed by I.
3. Conditional Composition: $\mathtt{T} = \mathtt{P}\langle v \rangle : \mathtt{Q}_v$. The process $\mathtt{T}$ behaves like the process $\mathtt{P}$ until that terminates. If $\mathtt{P}$ aborts, then $\mathtt{T}$ aborts. If $\mathtt{P}$ terminates normally, then the value v calculated by $\mathtt{P}$ is used to intialize the process $\mathtt{Q}$, and $\mathtt{T}$ then behaves like $\mathtt{Q}_v$.
4. Disabling Composition: $\mathtt{T} = \mathtt{P}\#\mathtt{Q}$. The process $\mathtt{T}$ behaves like the concurrent composition of $\mathtt{P}$ and $\mathtt{Q}$ until either terminates, then the other is aborted and $\mathtt{T}$ terminates. At most one process can stop; the remainder are aborted.
5. Synchronous Recurrent Composition: $\mathtt{T} = \mathtt{P}\langle v \rangle :: \mathtt{Q}_v$. This is a recursively defined as follows: $\mathtt{P} :: \mathtt{Q} = \mathtt{P} : (\mathtt{Q}; \mathtt{P} :: \mathtt{Q})$.
6. Asynchronous Recurrent Composition: $\mathtt{T} = \mathtt{P}\langle v \rangle :: \mathtt{Q}_v$. This is recursively defined as follows: $\mathtt{P} :: \mathtt{Q} = \mathtt{P} : (\mathtt{Q} \mid (\mathtt{P} :: \mathtt{Q}))$.

Operator Precedence: The operator precedence from loosest to tightest is as follows: Concurrent; Disabling; Sequential; Conditional; Synchronous Recurrent; Asynchronous Recurrent.

RS process composition operations are similar to the well-known CSP algebraic process model[13]. However, unlike CSP, in RS the notation can be seen as simply a shortcut for specifying automata; a process is a port automaton, and a process composition operation is two automata connected in a specific way. Composition operations include sequential, conditional and disabling compositions[9,10]. To analyze a network of processes, it is necessary to calculate how that network *changes* as time progresses and processes terminate and/or are created. This is the process-level equivalent of the PA transition function, combined with the axioms that define port-to-port communication. This *Process Transition function* can be used to analyze the behavior of RS networks.

RS has been used successfully in industrial settings for applications such as kit assembly. RS thus provides a mature system for robust, real-time distributed control. A short video showing RS controlling a robot arm and camera to track and grasp a moving object is available at http://csis.pace.edu/robotlab/clips/puma.avi.

3.2 RS and Soar

RS provides a powerful representational language for the system's dynamics, language and percepts; however, RS does not provide a mechanism for synthesizing the dynamics. Furthermore, RS lacks demonstrated cognitive plausibility, and in particular lacks a learning method.

We have implemented RS in Soar[8,12] to take advantage of Soar's cognitively plausible problem-solving and learning mechanisms. Soar uses *universal subgoaling* to organize its problem solving process into a hierarchy of subgoals, and uses *chunking* to speed and generalize that process. Universal subgoaling permits Soar to bring all its knowledge to bear on each subgoal. Chunking stores generalized preconditions for search control decisions, so that in future tasks similar search control decisions are made in a single step.

ADAPT's hierarchy of schemas is attached to the top-level state in Soar. ADAPT uses Soar's universal subgoaling to create its hierarchy of schemas, via operators that select, instantiate and interconnect schemas. This permits Soar's chunking method to learn how to build, monitor and modify networks of schemas.

General schemas are kept permanently in working memory, so that they function as a kind of declarative memory. Each schema is linked to other schemas that are relevant, e.g. tables are connected to chairs. This library of schemas in working memory that can be instantiated and activated is similar to the declarative memory in which ACT-R stores its chunks. We follow the example of ACT-R and permit ADAPT to create links between schemas to reflect relationships between them, such as “is a generalization of” and “is the opposite of”, as well as activation links labeled by weights that can be used by spreading activation in a manner similar to ACT-R.

Schemas have *input ports* and *output ports*. The ports of schema instances are connected to create ADAPT's network. Schemas can be decomposed into an assemblage of schemas that represent the world in more detail. Such *assemblage schemas* are attached to their parent schemas, creating a hierarchical structure that represents the world at varying granularities. Schemas that can be directly carried out by our Pioneer robot's Aria software possess an implementation attribute that contains the robot commands.

Each schema instance in the hierarchy has a weight and a priority, which are used by ADAPT to compute what the robot will actually do. Unlike Soar and other cognitive architectures, ADAPT does not select a single schema to execute on the robot. Instead, all active schemas that possess implementations are *blended* by ADAPT's *resolver* to create the actions actually executed on the robot platform. Blending involves weighting each schema implementation by its weight, adding together weighted implementations of equal priority, then disabling lower-priority actions if they contradict higher-priority actions.

For example, suppose a "move-forward" schema and a "turn-right" schema both have equal priority, with "move-forward" weighted by 0.9 and "turn-right" weighted by 0.1". The implementation of "move-forward" is for both wheels to move at speed 100, and the implementation of "turn-right" is for the right wheel not to move and the left wheel to move at speed 100. Then the blend of these two schemas will move the left wheel at speed 100 and the right wheel at speed 90, resulting in a slight angling to the right. If another move schema has a lower priority, it will be disabled. If another schema with a lower priority does not contradict the moves, e.g. a schema that tilts the camera, then it will be executed concurrently.

ADAPT subgoals in the same manner as Soar, but subgoaling does not necessarily choose one schema, but instead assigns weights for blending. Choosing one schema to execute is simply making one schema's weight 1 and all others' weights 0.

ADAPT's chunking mechanism is that of Soar. It creates a new rule (chunk) that summarizes preconditions for a subgoal's result. If the subgoal is selecting, instantiating or removing a schema, then the chunk will perform the operation in a single step. If the subgoal operates on multiple schemas, then the chunk will perform all the schema operations at once; in this way ADAPT can learn truly concurrent operations. For example, a subgoal may be to keep an object in focus while turning right. ADAPT can decide to instantiate a schema to turn the cameras left while it is running a schema to turn right. The decisions to instantiate and start both schemas will appear in one chunk, thus encoding distributed, concurrent control.

Thus, ADAPT's approach to modeling behaviors is to learn port automata from observed behaviors. The methods of learning automata are based on well-known methods of inductive inference of automata.

4. IMPLEMENTATION OVERVIEW

The overall implementation approach is that schemas are essentially declarative versions of Soar operators. The actual Soar operators act mostly as task-independent architectural operators, e.g. instantiating schemas, starting and pausing schemas. Task goals are persistent objectives; Soar goals are architectural goals, such as resolving impasses.

Schemas are proposed by Soar elaborations in the same way that Soar operators are. Proposing a schema attaches it to the top state with its parameters. Schemas are started by an operator that makes its ^status active. Schemas are also connected to other top-level schemas by Soar operators that connect input ports and output ports.

The following Soar proposal attaches the moving-around schema to the top state. It is initially inactive. A Soar operator must fire to make it active.

```
sp { propose*moving*around*schema
  (state <s> ^superstate nil)
  -->
  (<s> ^operator <o> + =)
  (<o> ^name start-schema ^schema <sc>)
  (<sc> ^name moving-around
       ^weight 100
       ^param.target-velocity 200)
}
```

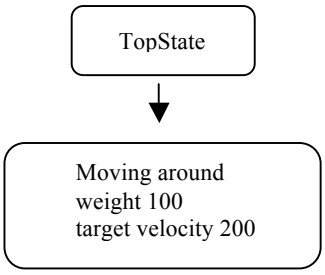

Figure 3. The moving around schema.

Assemblage subschemas are proposed by Soar elaborations, which attach them to the top state. Each assemblage subschema consists of a root schema and subschemas for the sensory, task and motor subschemas. The root schema points to its parent (the schema that it implements) and has ^status proposed. An operator named attach-assemblage makes the parent point to the assemblage's root and changes the assemblage's ^status to active. Doing it this way permits activation levels to be spread among assemblages to determine which one should be attached to the parent.

Currently, the only kind of assemblage subschemas available are task-unit schemas, which consists of three kinds of subschemas: sensory subschemas, task subschemas, and motor subschemas. The sensory schemas are typically connected to sensors, e.g. the sonars, and interpret the results, e.g. if something is close. The task schemas are the "program", i.e. they contain the logic that decides what to do. The motor schemas implement actions such as how to turn right.

In the picture below, there are four detect-nearby sensory schemas that implement a fuzzy rule for nearness of objects. Each of these schemas is attached to a sonar-reading sensory schema that reads the sonar sensors in one direction. The detect-nearby schemas inhibit the movement of the basic-moving-control schema (the small circles denote inhibitory connections.) The basic-moving-control schema is a task schema whose output controls the three motor schemas: turn-left, turn-right and move-forward.

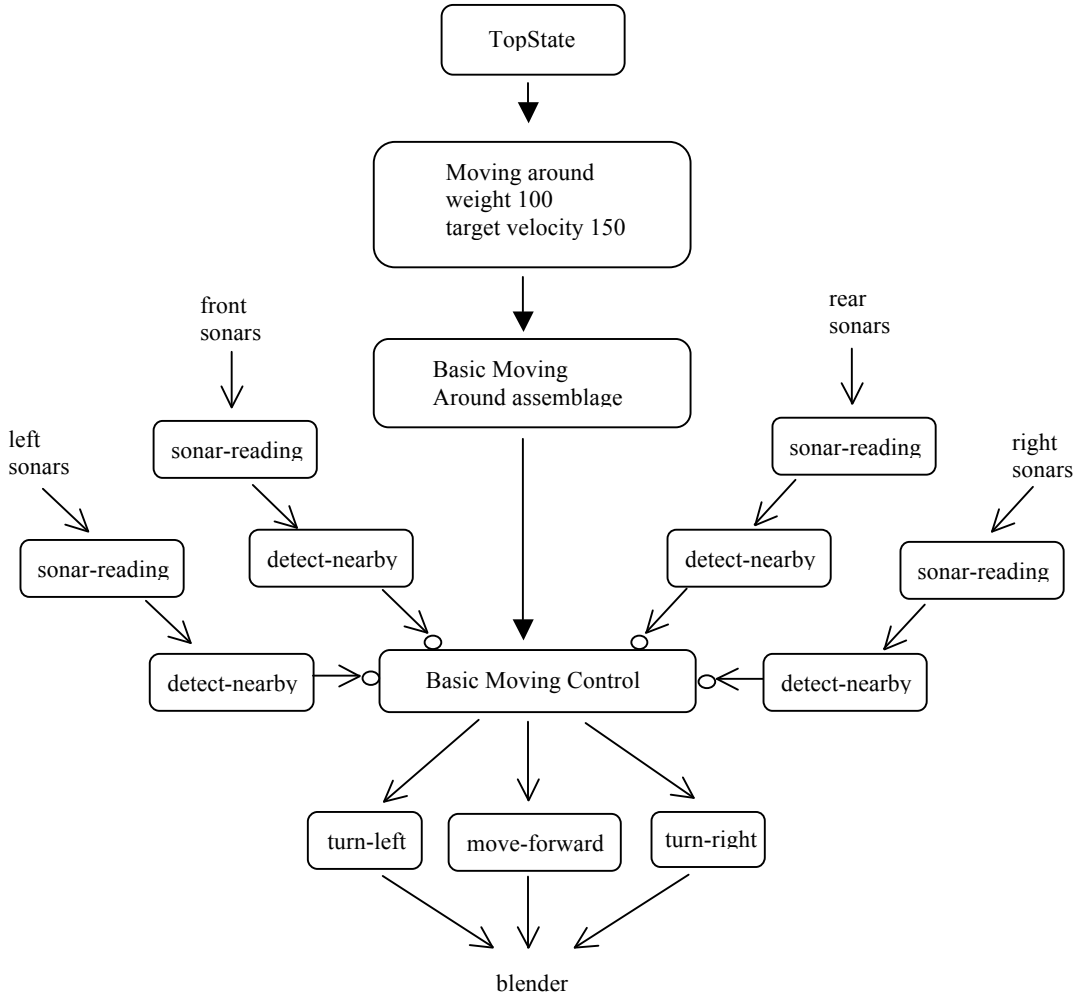

Figure 4. A simple network of schemas.

All schema port connections and internal computations are performed by Soar elaborations for maximal speed. Soar operators are used only for instantiating and starting schemas, stopping schemas, and the RS process composition operations. These Soar operators work, for the most part, by changing the ^status of schemas.

4. SUMMARY

To this point, most of the work has been on basic implementation. Getting all the software components to talk nicely to each other has been very hard. We have completed this implementation. ADAPT's inference engine can create basic entities in the virtual world in real time based on its vision data and update them to reflect new percepts as the robot moves. This has been implemented for a very small hand-coded library of known objects and their behaviors.

Also, we have demonstrated successfully that ADAPT can listen to a person, generate an appropriate response using a discourse model, and speak the response. The discourse models are also constructed by hand. Basic image schemas have

been implemented by hand and used to provide semantics for simple navigational concepts. We are currently expanding the libraries of objects and behaviors that can be modeled.

REFERENCES

[1] Arbib, M. A., 2003. "Schema theory", in The Handbook of Brain Theory and Neural Networks, Arbib, M. A., editor, 2nd Edition, MIT Press, 993-998 (2003).

[2] Benjamin, D. P., Lyons, D., and Lonsdale, D., "Embodying a Cognitive Model in a Mobile Robot", Proceedings of the SPIE Conference on Intelligent Robots and Computer Vision, Boston, (2006).

[3] Benjamin, D. P., Achtemichuk, T., and Lyons, D., "Obstacle Avoidance using Predictive Vision based on a Dynamic 3D World Model", Proceedings of the SPIE Conference on Intelligent Robots and Computer Vision, Boston, (2006).

[4] Benjamin, D. P., Lyons, D., and Lonsdale, D., "Designing a Robot Cognitive Architecture with Concurrency and Active Perception", Proceedings of the AAAI Fall Symposium on the Intersection of Cognitive Science and Robotics, Washington, D.C., (2004).

[5] Benjamin, D. P., Lyons, D., and Lonsdale, D., "ADAPT: A Cognitive Architecture for Robotics", Proceedings of the International Conference on Cognitive Modeling, Pittsburgh, Pa., (2004).

[6] Benjamin, D. P., "On the Emergence of Intelligent Global Behaviors from Simple Local Actions", Journal of Systems Science, special issue: Emergent Properties of Complex Systems, Vol. 31, No. 7, 861-872, (2000).

[7] Green, N. and Lehman, J.F., "An Integrated Discourse Recipe-Based Model for Task-Oriented Dialogue", Discourse Processes, 33(2), (2002).

[8] Laird, J.E., Newell, A. and Rosenbloom, P.S., "Soar: An Architecture for General Intelligence", Artificial Intelligence 33, 1-64, (1987).

[9] Lyons, D.M., "Representing and Analysing Action Plans as Networks of Concurrent Processes", IEEE Transactions on Robotics and Automation, (1993).

[10] Lyons, D.M. and Arbib, M.A., "A Formal Model of Computation for Sensory-based Robotics", IEEE Transactions on Robotics and Automation 5(3), (1989).

[11] Nelson, G., Lehman, J.F., and John, B.E., "Integrating cognitive capabilities in a real-time task," In Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society. Atlanta, GA, (1994).

[12] Newell, A., [Unified Theories of Cognition], Harvard University Press, Cambridge, Massachusetts, (1990).

[13] Schneider, S., [Concurrent and Real-time Systems: The CSP Approach], Wiley, (1999).

[14] Steenstrup, M., Arbib, M.A., Manes, E.G., "Port Automata and the Algebra of Concurrent Processes", JCSS 27(1), 29-50, (1983).